1)

T1 -> T2

R1(X), W1(X), R1(Y), W1(Y), R2(X), W2(X)

T2 -> T1

R2(X), W2(X), R1(X), W1(X), R1(Y), W1(Y)

There are 15 possible schedules for T1 and T2:

Conflict serializable schedules:

-same data item

-diff transactions

-write is one of them

R1(Y) W1(Y) R1(X) W1(X) R2(X) W2(X)

R1(X) R1(Y) W1(X) W1(Y) R2(X) W2(X)

R1(X) R1(Y) W1(Y) W1(X) R2(X) W2(X)

R1(Y) R1(X) W1(Y) W1(X) R2(X) W2(X)

R1(Y) R1(X) W1(X) W1(Y) R2(X) W2(X)

R1(X) W1(X) R2(X) W2(X) R1(Y) W1(Y)

R1(X) W1(X) R2(X) R1(Y) W1(Y) W2(X)

R2(X) W2(X) R1(X) R1(Y) W1(X) W1(Y)

R2(X) W2(X) R1(X) R1(Y) W1(Y) W1(X)

R2(X) W2(X) R1(Y) W1(Y) R1(X) W1(X)

R2(X) R1(Y) W2(X) R1(X) W1(X) W1(Y)

R2(X) R1(Y) W2(X) W1(Y) R1(X) W1(X)

R2(X) R1(Y) W1(Y) W2(X) R1(X) W1(X)

Not conflict serializable schedules:

R1(X) R2(X) W2(X) W1(X) R1(Y) W1(Y)

R1(X) R2(X) W2(X) R1(Y) W1(X) W1(Y)

R1(X) R2(X) W2(X) R1(Y) W1(Y) W1(X)

R1(X) R2(X) W1(X) R1(Y) W1(Y) W2(X)

R1(X) R2(X) W1(X) W2(X) R1(Y) W1(Y)

R1(X) R2(X) W1(X) R1(Y) W2(X) W1(Y)

R2(X) R1(X) W2(X) W1(X) W1(Y) R1(Y)

R2(X) R1(X) W2(X) W1(X) R1(Y) W1(Y)

R2(X) R1(X) W2(X) R1(Y) W1(X) W1(Y)

R2(X) R1(X) W1(X) W2(X) R1(Y) W1(Y)

R2(X) R1(X) W1(X) R1(Y) W1(Y) W2(X)

R2(X) R1(X) W1(X) R1(Y) W2(X) W1(Y)

2)

Schedule 1 is not conflict serializable since the schedule creates a cycle between

three transactions.

R1(x), R3(x), W1(x), R2(x), W3(x)

Schedule 2 is not conflict serializable since the schedule creates a cycle between

three transactions.

R1(x), R3(x), W3(x), W1(x), R2(x)

Schedule 3 is conflict serializable since the schedule does not a contain a cycle.

R3(x), R2(x), W3(x), R1(x), W1(x)

The equivalent conflict serial schedule is T2 -> T3 -> T1

3)

Precedence graph for schedule 1:

Precedence graph for schedule 2:

Precedence graph for schedule 3:

4)

|  |  |  |
| --- | --- | --- |
| Time | T1 | T2 |
| T1 | begin\_transaction |  |
| T2 | Write\_lock(bal x) | begin\_transaction |
| T3 | Read(bal x) | Write\_lock(bal y) |
| T4 | Bal x = bal x – 10 | Read (bal y) |
| T5 | Write (bal x) | Bal y = bal y + 100 |
| T6 | Write\_lock(bal y) | Write (bal y) |
| T7 | WAIT | Write\_lock(bal x) |
| T8 | WAIT | WAIT |
| T9 | WAIT | WAIT |
| T10 |  | WAIT |
| T11 |  |  |

An impasse that may result when two or more transactions are each waiting for locks to be released that are held by the other.

For the above schedule, the two transactions T1 and T2 are deadlocked because each is waiting for the other to release a lock on an item it holds. At time t2, transaction T1 requests and obtains an exclusive lock on item bal x and at time t3 transaction T2 obtains an exclusive lock on item bal y. Then at t6, T1 requests an exclusive lock on item bal y. Because T2 holds a lock on bal y, transaction T1 waits. Meanwhile, at time t7, T2 requests a lock on item bal x, which is held by transaction T1. Neither transaction can continue, because each is waiting for a lock it cannot obtain until the other completes. Once deadlock occurs, the applications involved cannot resolve the problem. Instead, the DBMS has to recognize that deadlock exists and break the deadlock in some way.

5)

6)

7) If in a particular schedule, failure of one transaction leads to series of rollbacks or aborts, then such a schedule is called as a Cascading Rollback. For example, transaction T1 writes uncommitted x that is read by transaction T2, and transaction T2 writes uncommitted x that is read by transaction T3. Suppose at this point T1 fails. So, T1 must be rolled back since T2 is dependent on T1, T2 must be rolled back, and since T3 is dependent on T2, T3 must be rolled back. Because of T1 rollback, all T2, T3 should be rollback. This singularity, in which a single transaction failure leads to a series of transaction rollbacks.